K-Nearest Neighbor Algorithm

Introduction

K-nearest neighbors (KNN) is a type of supervised learning algorithm used for both regression and classification. KNN tries to predict the correct class for the test data by calculating the distance between the test data and all the training points. Then select the K number of points which is closest to the test data. The KNN algorithm calculates the probability of the test data belonging to the classes of 'K' training data and class holds the highest probability will be selected. In the case of regression, the value is the mean of the 'K' selected training points.

What are K-Nearest Neighbors? Does it relate to my next door neighbor at all? KNN is a supervised learning algorithm used both as a classification and regression.

Imagine a circle, with you in the middle, now put around it another circle with your closest three friends, and a circle around that with less close five friends. Now think of two popular TV sitcoms, we will pick Friends and Fresh Prince in this particular case. What are your closest group of three friends likely to pick as their favourite show? And subsequently, the circle of five friends around that? As KNN is a distance based classifier, the more close two points are, the greater the similarities in behaviour and therefore selection choice. The different methods used to measure the distance are

- Manhattan
- Euclidean

Pros and Cons of using KNN

Pros:

- Simplistic algorithm — uses only value of K (odd number) and the distance function (Euclidean, as mentioned today).
- Efficient method for small datasets.

- Utilises “Lazy Learning.” In doing so, the training dataset is stored and is used only when making predictions therefore making it more quick than Support Vector Machines (SVMs) and Linear Regression.

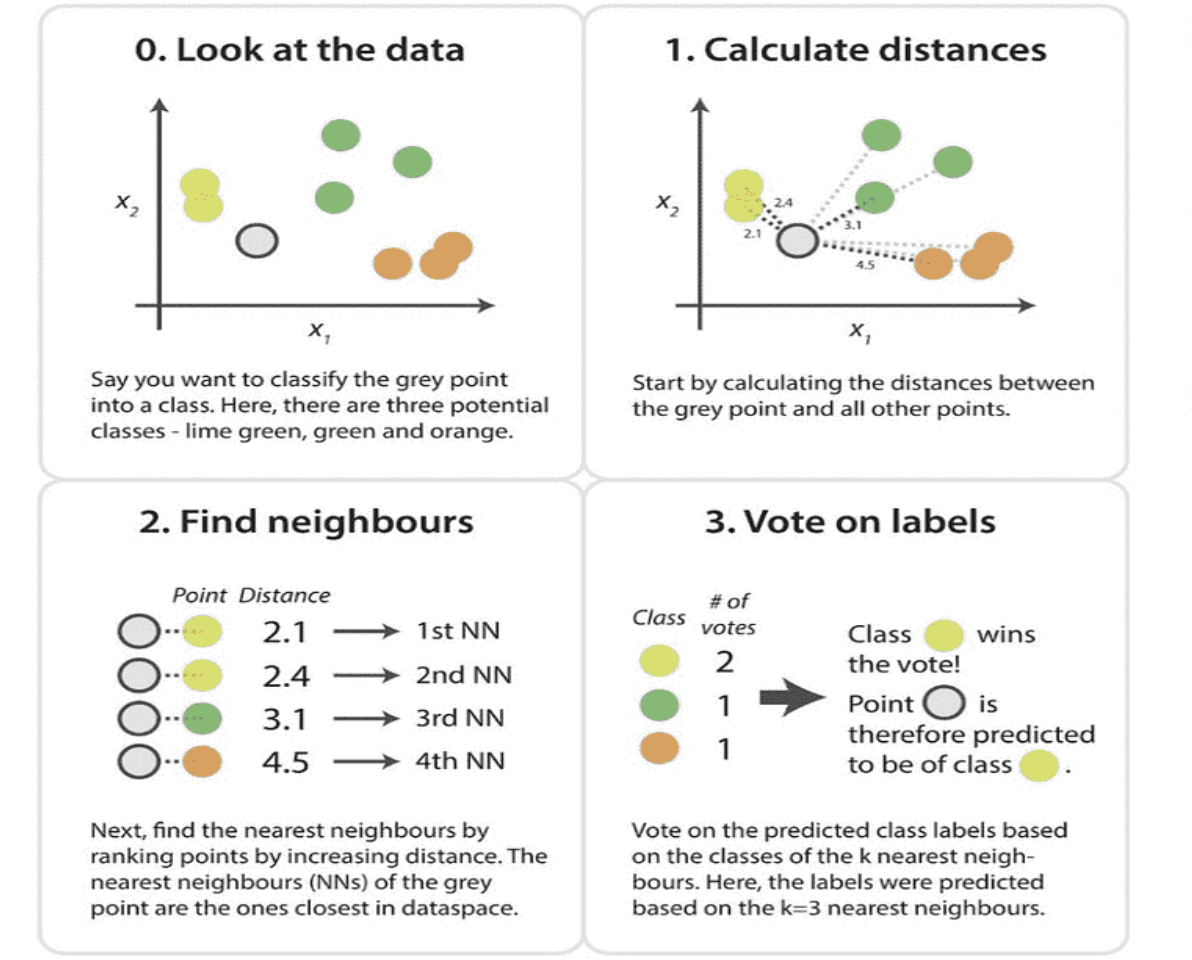
Cons:

- Large datasets take longer to process.
- Requires feature scaling, and inability to do will result in wrongful predictions.
- Noisy data can result in over-fitting or under-fitting of data.

Applications of KNN

1. Text mining
2. Agriculture
3. Finance
4. Medical
5. Facial recognition
6. Recommendation systems (Amazon, Hulu, Netflix, etc)

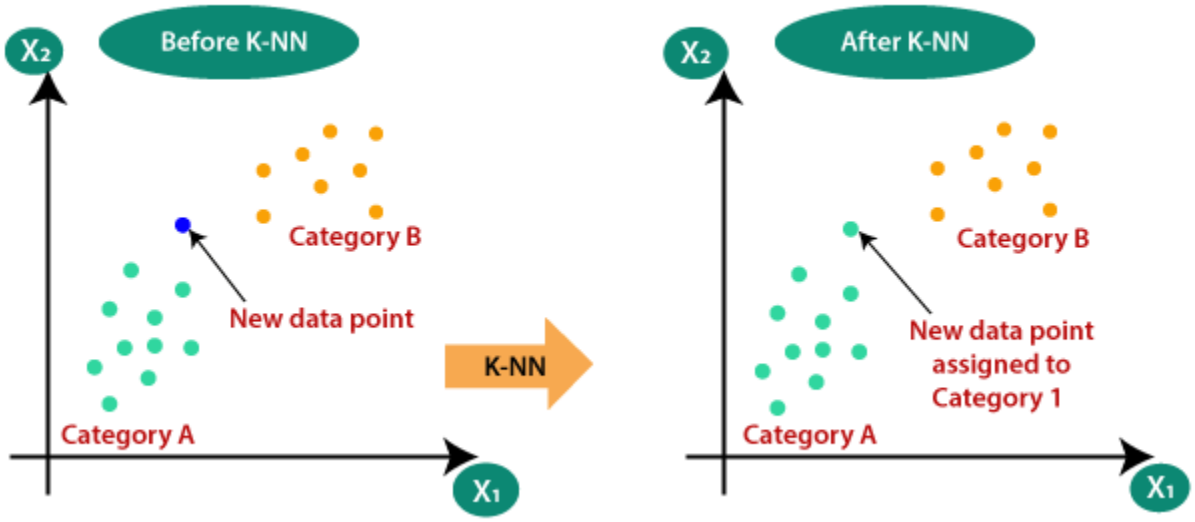
Let see the below example to make it a better understanding



Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

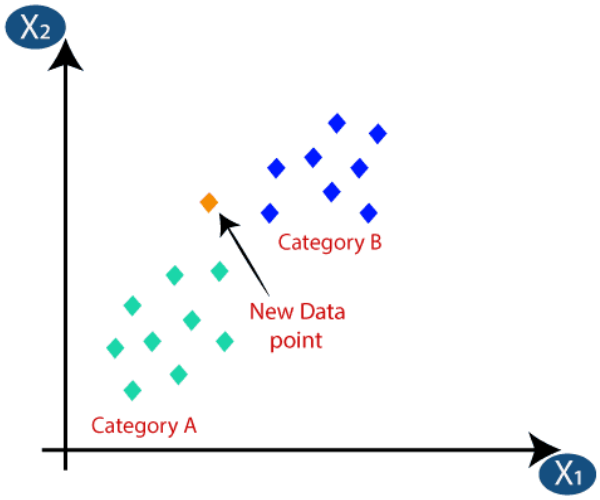


How does K-NN work?

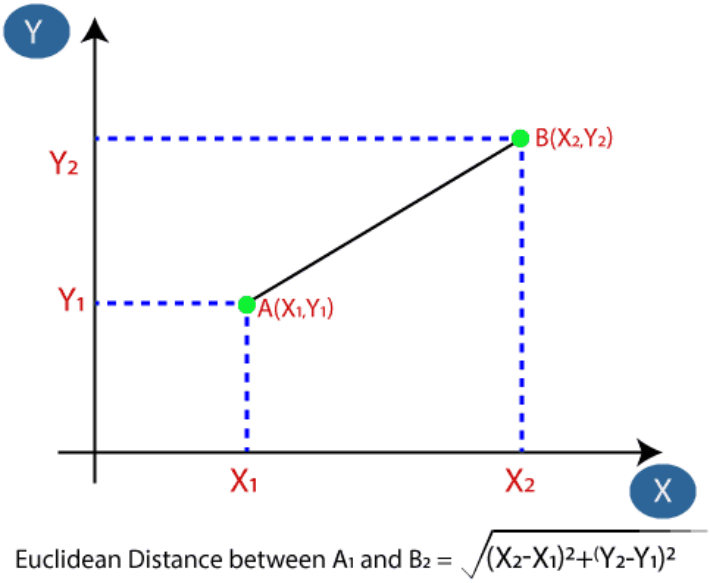
The K-NN working can be explained on the basis of the below algorithm:

- Step-1: Select the number K of the neighbors
- Step-2: Calculate the Euclidean distance of K number of neighbors
- Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.
- Step-4: Among these k neighbors, count the number of the data points in each category.
- Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.
- Step-6: Our model is ready.

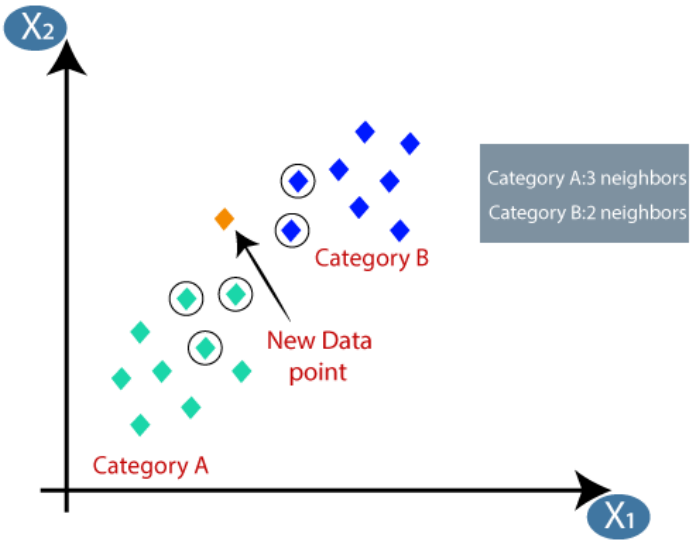
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the Euclidean distance between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



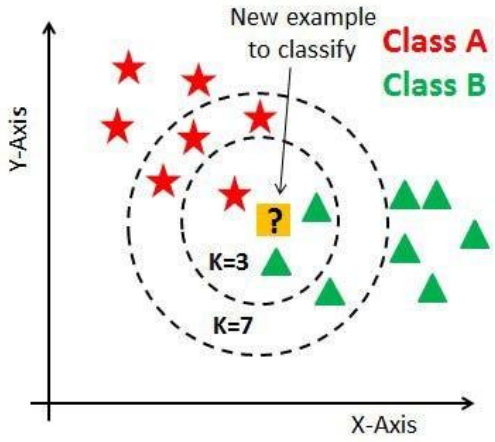
- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

How to choose a K value?

K value indicates the count of the nearest neighbors. We have to compute distances between test points and trained labels points. Updating distance metrics with every iteration is computationally expensive, and that's why KNN is a lazy learning algorithm.



- As you can verify from the above image, if we proceed with K=3, then we predict that test input belongs to class B, and if we continue with K=7, then we predict that test input belongs to class A.
- That's how you can imagine that the K value has a powerful effect on KNN performance.

Then how to select the optimal K value?

- There are no pre-defined statistical methods to find the most favorable value of K.
- Initialize a random K value and start computing.
- Choosing a small value of K leads to unstable decision boundaries.
- The substantial K value is better for classification as it leads to smoothening the decision boundaries.
- Derive a plot between error rate and K denoting values in a defined range. Then choose the K value as having a minimum error rate.

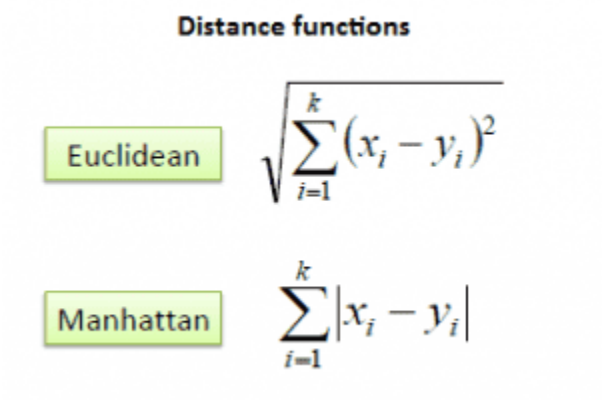
Now you will get the idea of choosing the optimal K value by implementing the model.

Calculating distance:

The first step is to calculate the distance between the new point and each training point. There are various methods for calculating this distance, of which the most commonly known methods are — Euclidian, Manhattan (for continuous) and Hamming distance (for categorical).

Euclidean Distance: Euclidean distance is calculated as the square root of the sum of the squared differences between a new point (x) and an existing point (y).

Manhattan Distance: This is the distance between real vectors using the sum of their absolute difference.



Hamming Distance: It is used for categorical variables. If the value (x) and the value (y) are the same, the distance D will be equal to 0 . Otherwise D=1.

$$D_H = \sum_{i=1}^{k}|x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

For a better understanding of how this can look like in a computer science topic, you can find below an HTML-code. A tree helps to structure a website and websites can normally be depicted using a tree.

Practical Implementation with Python

Now, let's dive into implementing KNN using Python. We'll walk through a step-by-step example of predicting diabetes based on health data.

```
# Importing necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, f1_score, accuracy_score

# Loading the dataset
dataset = pd.read_csv('diabetes_dataset.csv')

# Data preprocessing
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Fitting the KNN classifier to the training data
classifier = KNeighborsClassifier(n_neighbors=11, p=2)
classifier.fit(X_train, y_train)

# Predicting the test set results
y_pred = classifier.predict(X_test)

# Evaluating the model
conf_matrix = confusion_matrix(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

print("Confusion Matrix:\n", conf_matrix)
print("F1 Score:", f1)
print("Accuracy Score:", accuracy)
```

Conclusion

In conclusion, the K Nearest Neighbors algorithm is a versatile and powerful tool in the field of machine learning. Its simplicity, coupled with its effectiveness, makes it a popular choice for various classification tasks. By mastering KNN and its implementation in Python, you'll be equipped to tackle a wide range of real-world problems.